ROS in Autonomous Systems Applications
(Raspberry Pi / ROS Installation - Process)

This Document is prepared to show some experiments on a ROS-enabled Raspberry Pi Model 3 B+ connected with Arduino Uno

Table of Contents

ROS in Autonomous Systems Applications
(Raspberry Pi / ROS Installation - Process)

Introduction

We will work with Raspberry pi 3 Model B+ as the high-level embedded processor. Additionally, Arduino UNO will be used for the low-level processes.

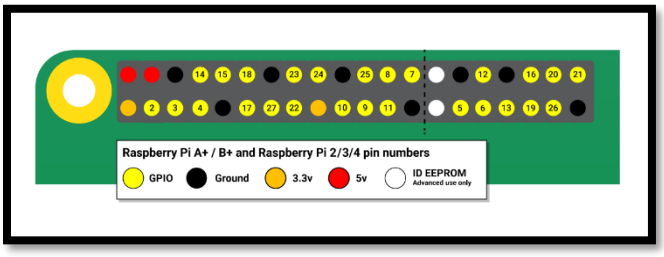
The programming language that will be used is python for the ROS-enabled Raspberry Pi.

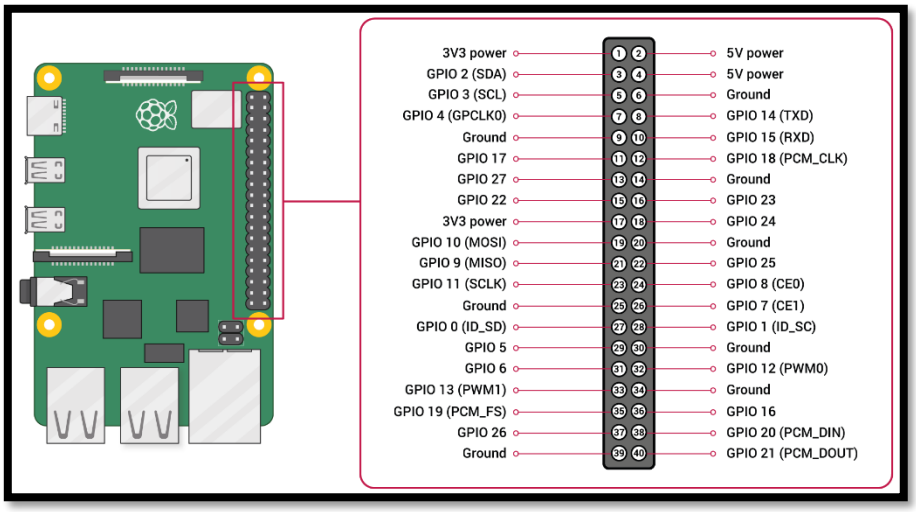
In this tutorial, basic experiments to test several functions will be explained. Some of these experiments are built using python compiler only. While some are using ROS nodes programmed by python as well. Moreover, basic experiments showing communication between the high- and low-level processors will be explained.

ROS in Autonomous Systems Applications
(Raspberry Pi / ROS Installation - Process)

Raspberry Pi Pins Layout

The figure below shows the pins layout. The pins are divided into 4 categories; General Purpose Input Output pins (GPIO), Ground, Voltages (+5 and +3.3 V), and ID EEPROM.



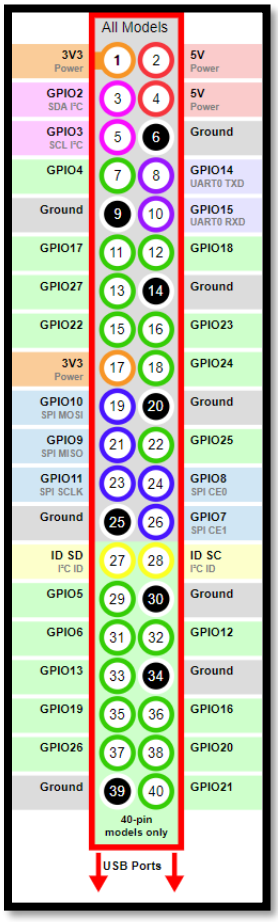


The GPIO pin designated as an **output** pin can be set to high (3V3) or low (0V) and as an **input** pin can be read as high (3V3) or low (0V).

This is made easier with the use of internal pull-up or pull-down resistors. Pins GPIO2 and GPIO3 have fixed pull-up resistors, but for other pins this can be configured in software. GPIO pins also can be used with a variety of alternative functions such as Pulse-Width Modulation (PWM) and communication protocols as SPI, I2C, and Serial.

ROS in Autonomous Systems Applications
(Raspberry Pi / ROS Installation - Process)

Pins Numbering



The main package that will be used in this experiment is ***gpiozero*** ***(***See the library's comprehensive documentation at https://gpiozero.readthedocs.io/en/stable/***)***.

This library uses Broadcom (BCM) pin numbering for the GPIO pins, as opposed to physical (BOARD) numbering. Unlike in the ***RPi.GPIO*** library, this is not configurable. However, translation from other schemes can be used by providing prefixes to pin numbers (see figure).

Any pin marked "GPIO" in the diagram can be used as a pin number. For example, if an LED was attached to "GPIO17" you would specify the pin number as 17 rather than 11.

If you wish to use physical (BOARD) numbering you can specify the pin number as "BOARD11". If you are familiar with the wiringPi pin numbers (another physical layout) you could use "WPI0" instead. Finally, you can specify pins as "header:number", e.g. "J8:11" meaning physical pin 11 on header J8 (the GPIO header on modern Pis). Hence, the following lines are all equivalent:

```
>>> led = LED(17)
>>> led = LED("GPIO17")
>>> led = LED("BCM17")
>>> led = LED("BOARD11")
>>> led = LED("WPI0")
>>> led = LED("J8:11")
```

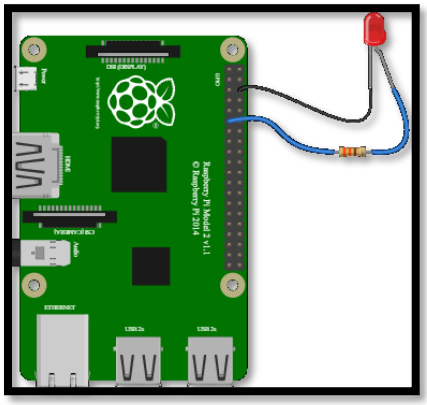
Experiments (1-2): Raspberry Pi

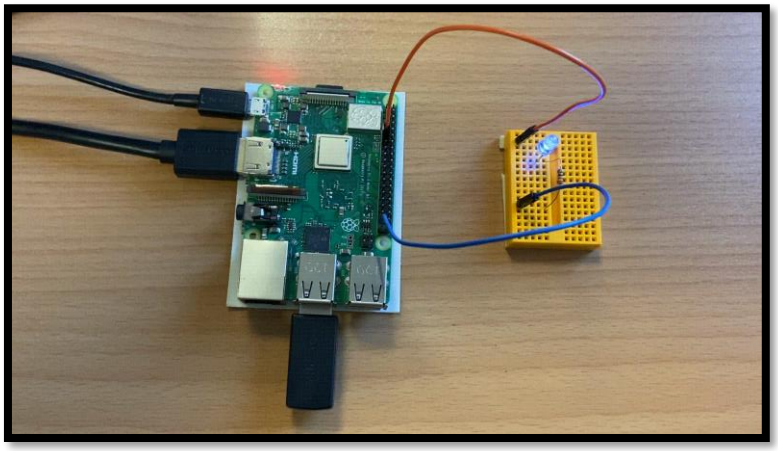
Experiment 1: Raspberry Pi - *(Blinking LED)*

The purpose of this experiment is to write a python code that allows a LED to blink on and off with sampling time of 1 second for 5 times.

ROS in Autonomous Systems Applications
(Raspberry Pi / ROS Installation - Process)

```
#!/usr/bin/env python

from gpiozero import LED
from time import sleep

led = LED(17)

for i in range (0,5):
 led.on()
 sleep(1)
 led.off()
 sleep(1)
```



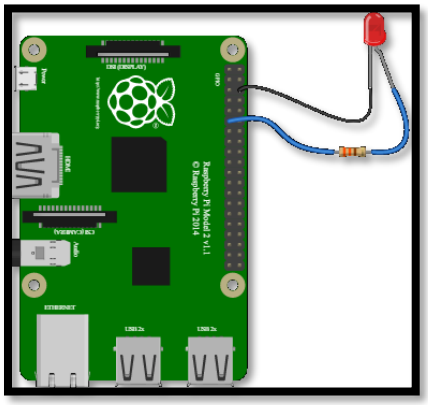


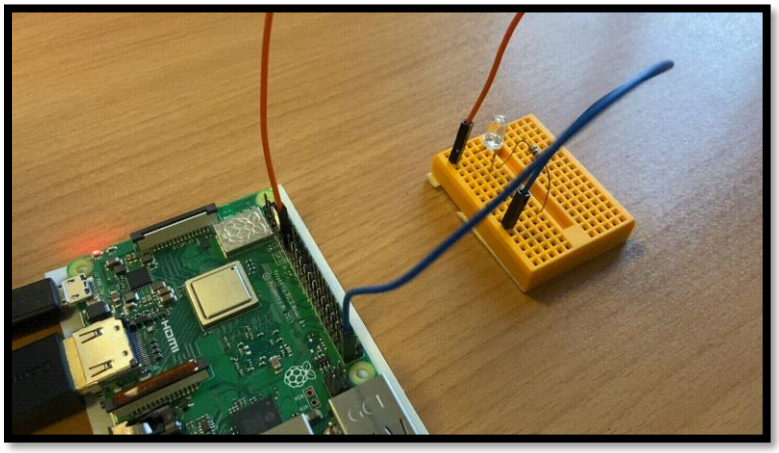
ROS in Autonomous Systems Applications
(Raspberry Pi / ROS Installation - Process)

Experiment 2: Raspberry Pi - *(PWM LED)*

The purpose of this experiment is to write a python code that allows a LED to have its brightness value set using PWM varying from 0 to 100% with sampling time of 0.05 second.

```
#!/usr/bin/env python

from gpiozero import PWMLED
from time import sleep

led = PWMLED(17)

for i in range (0,256):
 led.value = float(i)/256
 sleep(0.05)
```





Experiments (3-6): ROS/Raspberry Pi

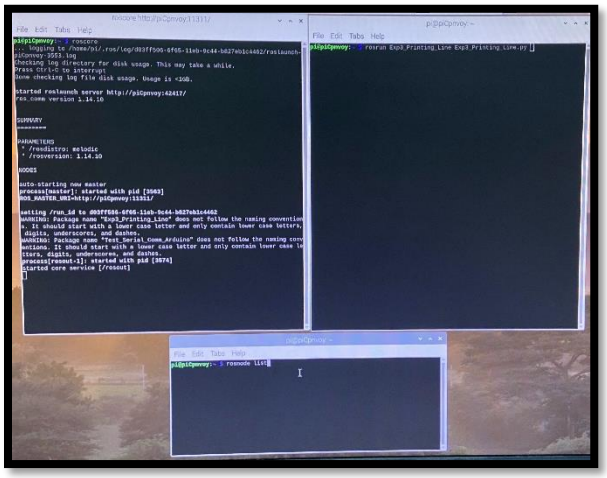
For the upcoming experiments between ROS and Raspberry Pi, a ROS package is created named `Exp_ROS_RaspberryPi` (*Follow steps in the ROS tutorials to create package in the catkin workspace*) with rospy and std_msgs dependencies and build the catkin_ws. Then, create a ROS python node (*Set access permission to Owner*) with the below code.

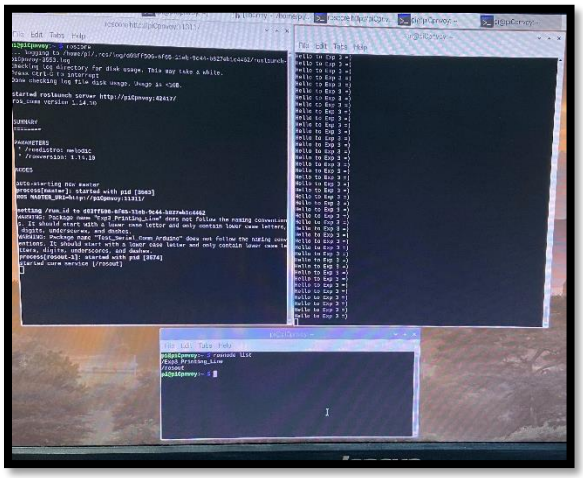
ROS in Autonomous Systems Applications
(Raspberry Pi / ROS Installation - Process)

Experiment 3: ROS/Raspberry Pi - *(Printing Line)*

The purpose of this experiment is to write a python code creating a ROS node called **`Exp3_Printing_Line`**. This node prints a line **`Hello to Exp 3 =)`** to the terminal as long as the node is activated till it is shutdown. (Show the activated nodes)

```
#!/usr/bin/env python

import rospy

rospy.init_node('Exp3_Printing_Line')

while not rospy.is_shutdown():
 print('Hello to Exp 3 =)')
```





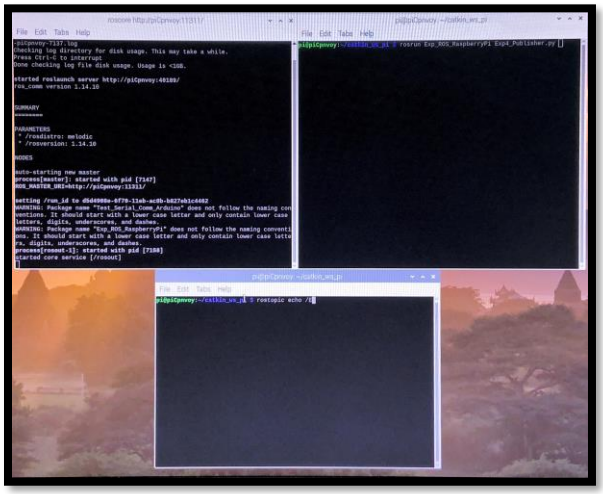
ROS in Autonomous Systems Applications
(Raspberry Pi / ROS Installation - Process)

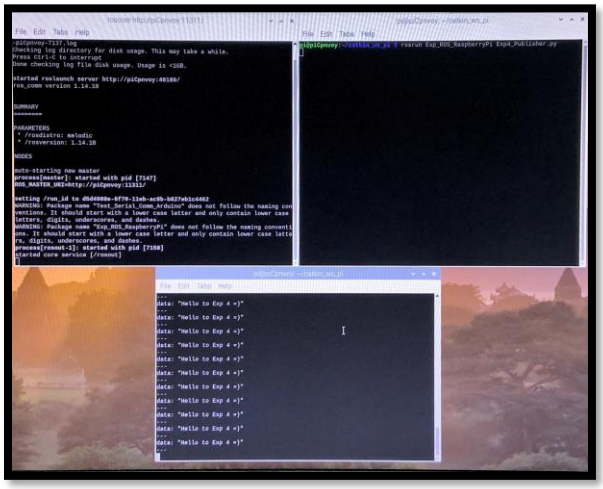
Experiment 4: ROS/Raspberry Pi - *(Publisher)*

The purpose of this experiment is to write a python code creating a **publisher** ROS node called **`Exp4_Publisher`**. This node publishes a string topic **`Hello to Exp 4 =)`** to be subscribed and echoed in the terminal as long as the node is activated till it is shutdown. (Show the activated nodes, topics, and messages)

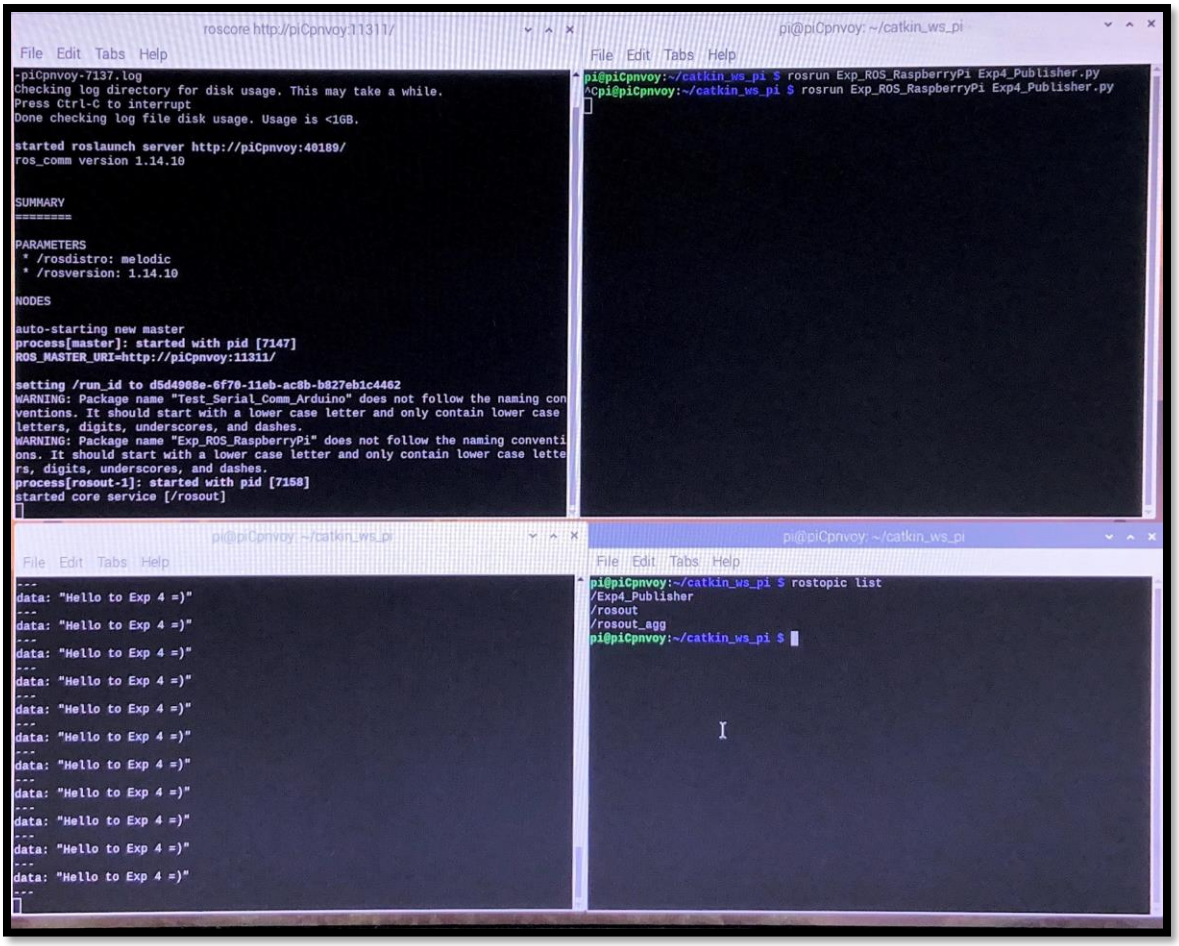
Publisher Node

```
#!/usr/bin/env python

import rospy
from std_msgs import String

rospy.init_node('Exp4_Publisher')
pub = rospy.Publisher('Exp4_Publisher', String, queue_size = 10)
rate = rospy.Rate(10)

while not rospy.is_shutdown():
 pub.publish('Hello to Exp 4 =)')
 rate.sleep()
```





ROS in Autonomous Systems Applications
(Raspberry Pi / ROS Installation - Process)



ROS in Autonomous Systems Applications
(Raspberry Pi / ROS Installation - Process)

Experiment 5: ROS/Raspberry Pi - *(Subscriber)*

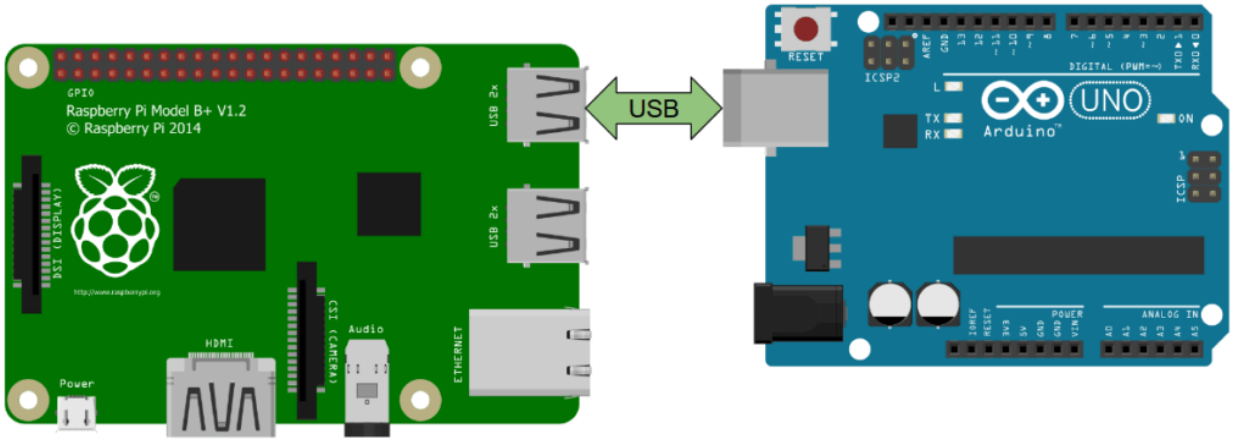
The purpose of this experiment is to write a python code creating a Subscriber ROS node called **Exp5_Subscriber**. This node subscribes and prints the string published topic by the publisher node (*previously created in Exp 4*) as long as the node is activated till it is shutdown. (Show the activated nodes, topics, and messages)

Subscriber Node

```
#!/usr/bin/env python

import rospy
from std_msgs import String

def callback(data):
 ToBePrinted = 'Received Msg is: '+ data.data
 Print(ToBePrinted)

rospy.init_node('Exp5_Subscriber')
subs = rospy.Subscriber('/Exp4_Publisher', String, callback)
rate = rospy.Rate(10)

while not rospy.is_shutdown():
 rospy.spin()
```

ROS in Autonomous Systems Applications
(Raspberry Pi / ROS Installation - Process)

Experiment 6: ROS/Raspberry Pi – Arduino - *(Serial Communication)*



ROS in Autonomous Systems Applications
(Raspberry Pi / ROS Installation - Process)

Experiment 6-1: ROS/Raspberry Pi *(Read)* – Arduino *(Write)* - *(PWM LED)*

The purpose of this experiment is to write a python code creating a ROS node called `Exp6_1_Serial_Comm_Arduino_Read`. This node reads the data sent by the Arduino through serial communication represented by the USB cable. The received data is sent as a LED brightness value set using PWM varying from 0 to 100%. The node keeps running as long as the node is activated till it is shutdown. (Show the activated nodes, topics, and messages).

The set sampling time is 0.05 second and baud rate of 9600.

First run on the terminal the command (*ls /dev/tty**) and observe the ACM port number (Ex. */dev/ttyACM0*)

Raspberry Pi ROS Node (*Read*)

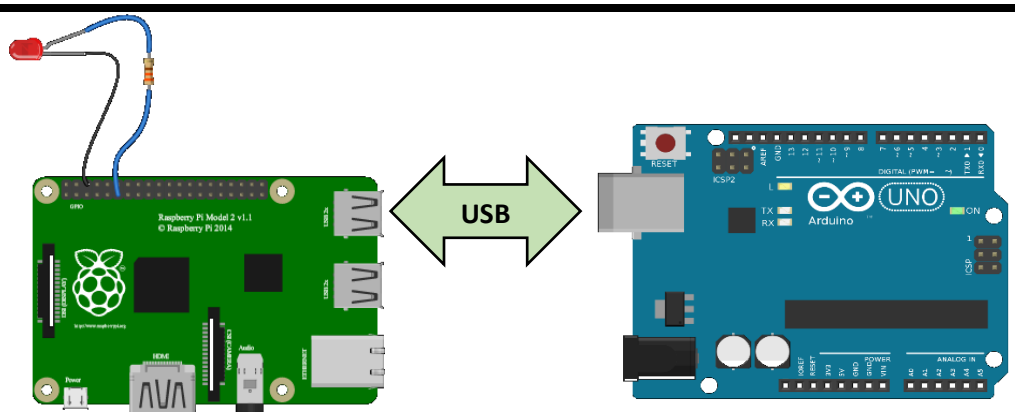
```
#!/usr/bin/env python

import rospy
import serial
from gpiozero import PWMLED
from time import sleep

rospy.init_node('Exp6_1_Serial_Comm_Arduino_Read')
rate = rospy.Rate(20)

ser = serial.Serial('/dev/ttyACM0',9600)
s = 0
led = PWMLED(17)

while not rospy.is_shutdown():
 read_serial = ser.readline()
 s = str(int(read_serial,16))
 led.value = float(s)/256
 rate.sleep()
```

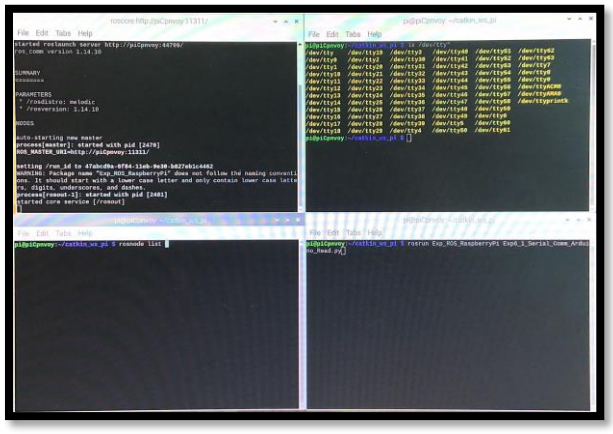
Arduino IDE (*Write*)

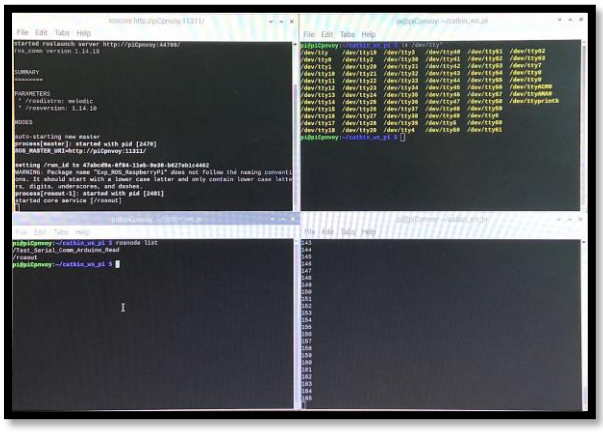
```
char dataString[50] = {0};
int a = 0;

void setup() {
 Serial.begin(9600);
```

ROS in Autonomous Systems Applications
(Raspberry Pi / ROS Installation - Process)

```
}

void loop() {
  sprintf(dataString,"%02X",a);
  Serial.println(dataString);
  a++;
  if(a>256){
    a = 0;
  }
  delay(50);
}
```







ROS in Autonomous Systems Applications
(Raspberry Pi / ROS Installation - Process)

ROS in Autonomous Systems Applications
(Raspberry Pi / ROS Installation - Process)

Experiment 6-2: ROS/Raspberry Pi *(Write)* – Arduino *(Read)* - *(PWM LED)*

The purpose of this experiment is to write a python code creating a ROS node called `Exp6_2_Serial_Comm_Arduino_Write`. This node writes a LED brightness value set using PWM varying from 0 to 100% to be read by Arduino through serial communication represented by the USB cable. The node keeps running as long as the node is activated till it is shutdown. (Show the activated nodes, topics, and messages).

The set sampling time is 0.05 second and baud rate of 9600.

First run on the terminal the command (*ls /dev/tty**) and observe the ACM port number (Ex. */dev/ttyACM0*)

Raspberry Pi ROS Node (*Write*)

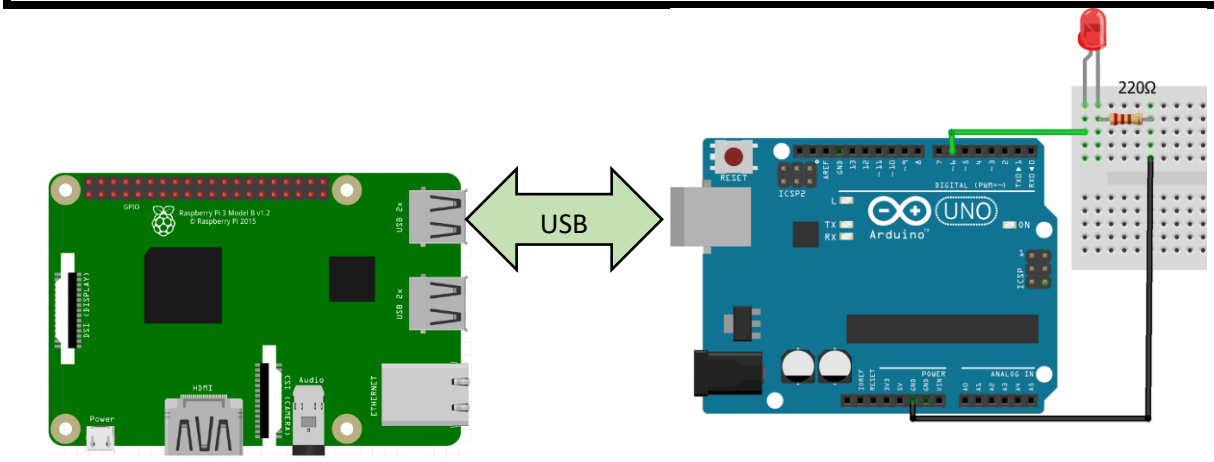
```
#!/usr/bin/env python

import rospy
import serial

rospy.init_node('Exp6_2_Serial_Comm_Arduino_Write')
rate = rospy.Rate(20)

ser = serial.Serial('/dev/ttyACM0',9600)
i = 0

while not rospy.is_shutdown():
 val = str(i) + '\n'
 ser.write(val)
 i += 1
 if i > 256:
  i = 0
 rate.sleep()
```

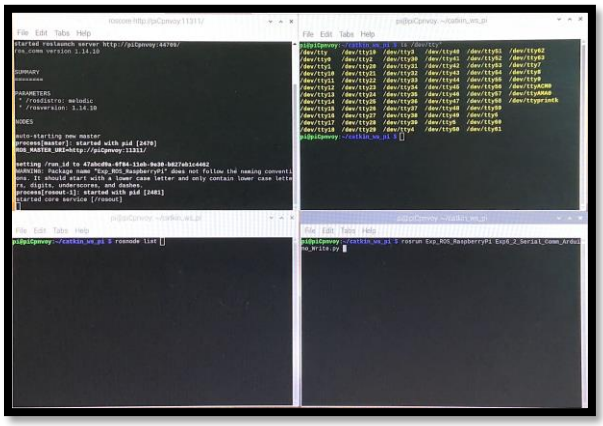
Arduino IDE (*Read*)

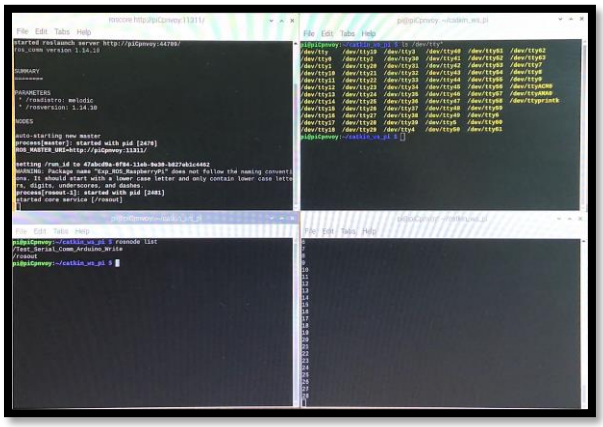
```
int led_pin = 6;

void setup() {
  Serial.begin(9600); // opens serial port, sets data rate to 9600 bps
  pinMode(led_pin, OUTPUT);
```

ROS in Autonomous Systems Applications
(Raspberry Pi / ROS Installation - Process)

```
}

void loop() {
  // send data only when you receive data:
 if (Serial.available() > 0) {
    String incomingByte = Serial.readStringUntil('\n'); // read the
incoming byte:
    int x = incomingByte.toInt();
    analogWrite(led_pin,x);
    delay(50);
  }
}
```







ROS in Autonomous Systems Applications
(Raspberry Pi / ROS Installation - Process)